Actividad 1.1 Implementación de la técnica de programación "divide y vencerás"

Tomas Orozco // A01562785
Adrian Chavez // A01568679

Tecnológico de Monterrey
Análisis y diseño de algoritmos avanzados (Gpo 600)

28 de agosto del 2023.
Chihuahua, Chihuahua, México.

Mergesort:

(algoritmo de ordenamiento)

Merge Sort es un algoritmo de ordenación eficiente que sigue el enfoque "divide y conquistarás". Funciona dividiendo la lista no ordenada en mitades más pequeñas, ordenando cada mitad de forma recursiva y luego fusionando las mitades ordenadas para obtener la lista final ordenada.

complejidad:

En el peor, mejor y caso promedio, Merge Sort tiene una complejidad de tiempo de O (n log n), donde 'n' es el número de elementos en la lista. Esto lo hace muy eficiente para listas grandes.

Casos de prueba:

- **Caso 1:**

En el primer caso decidimos probar con una lista completamente ordenada la cual es la mejor complejidad de este algoritmo.

- **Caso 2:**

En el segundo caso decimos probar con una lista completamente desordenada para probar con su peor caso y ver cómo funciona.

- **Caso 3:**

En este caso usaremos una lista muy corta para ver si esta afecta el rendimiento de este.

- **Caso 4:**

En este caso usaremos una lista bastante larga que permite ver si hay algún cambio dependiendo de lo largo de la lista.

Codigo:

```
int main()
{
    vector<double> input1{ 5,2,9,1,7,4,6 }; //Caso de prueba 1: Lista longitud media completamente desordenada
    vector<double> input2{ 7,6,5,4,3,2,1}; //Caso de prueba 2: Lista longitud media completamente ordenada
    vector<double> input3{ 15,10,25,5,20,30,35,40,45,1,50,2 }; //Caso de prueba 3: Lista longitud larga medio desordenada
    vector<double> input4{ 8,3,6,1,4 }; //Caso de prueba 4: Lista longitud chica medio desordenada

    vector<double> expected1{ 9,7,6,5,4,2,1 };
    vector<double> expected2{ 7,6,5,4,3,2,1 };
    vector<double> expected3{ 50,45,40,35,30,25,20,15,10,5,2,1 };
    vector<double> expected4{ 8,6,4,3,1};

    vector<vector<double>> inputs{ input1, input2, input3, input4 };
    vector<vector<double>> expectedOuts{ expected1, expected2, expected3, expected4 };

    cout << "--- MERGE SORT ---" << endl;

    for (int i = 0; i < inputs.size(); i++) {
        cout << "CASO " << i + 1<< ":" << endl;
        cout << "Input: ";
        printVect(inputs[i]);
        cout << "\nExpected Output: ";
        printVect(expectedOuts[i]);
        vector<double> res = mergesort(inputs[i], inputs[i].size()); //Llama a mergesort()
        if (expectedOuts[i] == res) {
            cout << "\nActual Output: ";
            printVect(res);
            cout << "\n- PRUEBA EXITOSA -" << endl;
        }
        else {
            cout << "\nActual Output: ";
            printVect(res);
            cout << "\n- PRUEBA FALLIDA - " << endl;
        }
        cout << "\n------------------------------" << endl;
    }

}
```

En esta parte del código se puede observar que tenemos los casos de prueba definidos en un vector y también lo que se espera que salga al final del ordenamiento el for de abajo checa si la prueba es exitosa o fallida después de llamar al mergesort.

```cpp
vector<double> mergesort(vector<double> a, int n) {  //T(n) = O(n*log n) En el mejor y peor de los casos
    if (n == 1) {
        return a;
    }

    int middle = n / 2;

    vector<double> arregloIzq;
    vector<double> arregloDer;

    for (int i = 0; i < middle; i++)
        arregloIzq.push_back(a[i]);
    for (int j = 0; j < n-middle ; j++)
        arregloDer.push_back(a[middle + j]);
                                                        //Función que divide
    arregloIzq = mergesort(arregloIzq, arregloIzq.size());  //# veces: n/2
    arregloDer = mergesort(arregloDer, arregloDer.size());  //# veces: n/2
                                                       //Función que ordena
    return merge(arregloIzq, arregloDer);              //# veces: n

}
```

En esta sección se hace la división y del vector y se declara que parte es izquierda y cual es derecha haciendo 2 vectores después de aquí empezaremos a ordenar en la siguiente parte del código.

```cpp
vector <double> merge(vector<double> a, vector<double> b) {
    vector<double> c;
    while (a.size() > 0 && b.size() > 0) {
        if (a[0] < b[0]) {
            c.push_back(b[0]);
            b.erase(b.begin());
        }
        else {
            c.push_back(a[0]);
            a.erase(a.begin());
        }
    }

    while (a.size() > 0) {
        c.push_back(a[0]);
        a.erase(a.begin());
    }

    while (b.size() > 0) {
        c.push_back(b[0]);
        b.erase(b.begin());
    }

    return c;
}
```

Aquí es donde se ordena las divisiones para darnos un orden de mayor a menor estas 2 funciones se mandan a la primera sección para ver si se ordenó bien.

Video: https://youtu.be/y52cdReJRQo?si=sGO19crOo_I6CRVy